Jr. of Industrial Pollution Control 33(2)(2017) pp 1745-1753
www.icontrolpollution.com
Research Article

CRITICALITY-AWARE PARTITIONED TASK SCHEDULING WITH LIMITED MIGRATORY ON MULTICORE PLATFORMS

NAGALAKSHMI K[1]* AND GOMATHI N[1]

[1]Department of Information Technology E.G.S. Pillay Engineering College Nagapattinam, Tamilnadu, India.

[2]Department of Computer Science and Engineering Vel Tech Dr.RR & Dr.SR University, Chennai, Tamilnadu, India.

(Received 25 May, 2017; accepted 22 December, 2017)

Key words: Mixed-criticality, Multicore processor, Task scheduling, Schedulability, Sporadic task

ABSTRACT

Mixed-critical (MC) systems, in which different functionalities of varying criticality levels may consolidate on a shared embedded platform, are an active area of research in safety-related environments. With the proliferation of MC system, the multicore processor is becoming the obvious design choice in current and future safety-critical domains. The real-time scheduling of certifiable MC systems on a multicore platform has been recognized as a great challenging issue, where using conventional scheduling algorithms may cause significant under-utilization of the platform's resources. In this work, we address this important dispute by proposing an effective optimal partitioning approach, the Criticality-aware Partitioned Algorithm (CaPA), that enables a limited number of migration of low-criticality workloads to improve the effectiveness of the schedulability by integrating the potential benefits of partitioned scheduling approaches. The results from extensive simulation under different situations demonstrate that CaPA always significantly outperforms existing MC partitioning heuristics in terms of acceptance ratios.

INTRODUCTION

The recent advances in CMOS scaling down technology enable processor manufacturers to fabricate more computational elements (cores) on a single-chip to realize high performance and reliability at low-cost. The implementation of multicore processors for an MC system will be pressurized by the ever-increasing demand for processing power, development cost, and by SWaP (Size, Weight, and Power) requirements. This has resulted in the integration of multiple tasks with varying criticalities onto a common platform. Regrettably, such consolidation may cause asymmetric inter-task interference effects (i.e., schedule disruptions) between various criticality levels that in turn results in poor processor utilization. In order to authenticate the timing correctness of system components on the different level of rigorousness, MC systems are subject to certifications.

A criticality level is defined as a degree of guarantee required against failure. A higher criticality level assigned to a task reveals that the higher degree of guarantee is required about the correctness of the task (workload). For validating the correctness of safety domain on each criticality level, their workloads are subject to certification requirements by different Certification Authorities (CAs). To validate the correctness of system behavior, such authorities often mandate conservative assumptions about the worst-case execution of the applications; these assumptions are usually far more pessimistic than the assumptions that the system manufacturer would use during all the phases of designing, implementing, and testing. Nevertheless, while CA is only involved in validating the safety-related applications of the system the system architect is responsible for guaranteeing that the whole system is correct, including the non-critical parts.

***Corresponding authors email: nagalaxmi@gmail.com; gomathin@veltechuniv.edu.in**

The introduction of concurrently executed applications with different criticalities into the engineering of MC subsystem bringing up new scheduling challenges which are not effectively tackled by traditional scheduling approaches. Recently, there is an increasing trend to propose novel MC scheduling approaches which are projected to carry out effective system utilization while ensuring timing guarantees and resource/ temporal separation among different workloads. In most proposed solutions, two criticality levels are defined: high and low. High-critical tasks (HCTs) are usually hard real-time workloads and their timing constraints always be satisfied even under all worst-case circumstances proposed by the certification authority. Low-critical tasks (LCTs) are soft real-time workloads because they can tolerate a certain amount of deadline misses.

Task scheduling approaches on multicore can be classified into two categories: partitioned scheduling (PS) (Andersson, *et al*., 2001), and the global scheduling (GS) approach (Andersson, 2008). In the PS, each workload is mapped to a designated core (processor). Each job from the same workload will be performed only on that specified core. In GS, all instances from various workloads assembled into a common queue, and hence each workload can be performed on any core. GS can provide maximum utilization, while PS enables a convenient and more deterministic implementation, which makes it a better choice for hard real-time workloads. Of late, a novel strategy, named semi-partitioned technique (Anderson, *et al*., 2005), has been developed. In this strategy, most workloads are allocated to their designated cores (i.e., similar to PS). But, some workloads are permitted to decompose into many subtasks and each subtask is allocated across various available cores. These workloads can also migrate across various cores when needed.

In this paper, we follow the latest MC scheduling approach (Burns and Baruah, 2013) that attempts to enable a limited service level for LCTs in the critical mode, instead of simply rejecting them (as in most other works like (Baruah, *et al*., 2011)). We develop a scheduling algorithm for an MC system on multicore processors. Since the MC system comprises critical workloads, a scheduling approach that preserves the predictability will be essential for these workloads. Nevertheless, PS will have an adverse effect on the overall system utilization because some cores will have idle capacity. This problem is even worse for MC workloads as they have different WCET (and therefore different utilizations) at the various level of assurance. A partitioning apt for one criticality might not be appropriate for other levels. To enable a deterministic and effective partitioning, we develop a Criticality-aware Partitioned Scheduling Algorithm (CaPA). In our algorithm, HCTs are statically allocated to cores while LCTs are allocated with some migration to exploit the cores effectively. The admitted task set will be partitioned completely until it continues in a normal execution mode. However, during mode transition, LCTs can be transferred to another core when needed. Experiments reveal that the CaPA outperforms other PS approaches over a range of parameters.

The organization of this article is as follows: Section II reviews some prior investigations which match our analysis. The sporadic task system considered in this paper is formally defined in Section III. Our proposed Criticality-aware Partitioned Scheduling Algorithm and the schedulability tests are given in Section IV. An evaluation is presented in Section V. We conclude this work in Section VI.

RELATED WORK

Numerous MC scheduling approaches in the context of multicore have been developed recently. One of the most commonly used MC scheduling approaches is the Adaptive Mixed Criticality (AMC) strategy developed by (Baruah, *et al*., 2011). In this model, all LCTs are overloaded in the critical mode (C-Mode). While this model provides timeliness guarantees for HCTs, no assurances are provided for LCTs in C-Mode, which is not acceptable for several real-time scenarios. For instance, in the control system of an unmanned aerial vehicle, sporadic delays can lead unacceptable performance degradation and uncertainty in the system behavior (Yip, *et al*., 2014). To relax the assumptions of AMC and ensure certain service level to LCTs, (Burns and Baruah, 2014) develop a model with an increasing period of LCTs in C-Mode, similar to Elastic Mixed-Criticality task model (E-MC) (Buttazzo, *et al*., 1998).

Su et al. take E-MC into account, and investigate Earliest Deadline First (EDF) based scheduling for single-core and multicore systems (Su and Zhu, 2013). In our work, we consider E-MC by enabling a limited number of LCTs to execute in C-Mode. In this work, we concentrate on fixed priority schemes, therefore, the E-MC system is used for static-priority scheduling, rather than EDF approach as in (Su and Zhu, 2013; Su, *et al*., 2013). One of the first works on the multicore scheduling of the MC system is suggested by Mollison et al. The authors implement five different criticality levels from A to E using

different scheduling policies (such as partitioned EDF, global EDF, cyclic executive and global best effort). A two level compositional technique is employed for scheduling. This algorithm also introduced a certain limitation on workloads such as demanding all inter-task arrival time of level B workloads be integer multiples of the level-A hyper period (Mollison, *et al*., 2010).

Baruah et al. evaluate PS and GS approach for an MC system and determined that PS is more efficient (Baruah, *et al*., 2014). This is because the selected utilization limits in schedulability analysis for GS are more conservative whereas PS can exploit more precise methods for schedulability tests (Baruah, *et al*., 2014). Furthermore, PS is implemented in industrial standards for MC scheduling (Rodriguez, *et al*., 2013). Therefore, in this work we concentrate on PS.

Gu et al. develop a PS approach called Mixed-criticality Partitioning with Virtual Deadlines (MPVD) (Gu, *et al*., 2014). In MPVD, HCTs are mapped by means of the Worst-Fit approach then LCTs are mapped by means of First-Fit. Workloads are ordered by utilization at their criticalities. Rodriguez *et al.* propose an analogous approach trying other combinations of mapping approaches (Best-Fit (BF), Worst-Fit (WF), First-Fit (FF) and Next-Fit (NF) (Rodriguez, *et al*., 2013). The authors conclude that an approach that allocates HCTs first using WF then LCTs using FF, both ordered by decreasing utilization achieves higher success rate. (Kelly, *et al*., 2011) compare different mapping heuristics for PS of the mixed-criticality system. Workloads are sorted by non-increasing utilization or by non-increasing criticality. They determined that FF and BF with decreasing criticality return the best performance.

PROBLEM DEFINITION AND NOTATION

In this section, we consider a set of n sovereign MC sporadic tasks $\tau = \{\tau_1^{£}, \tau_2^{£},...,\tau_n^{£}\}$, and a processor Ψ with λ cores $\Psi = \{\Psi_1,\Psi_2,...,\Psi_\lambda\}$. Each workload τ_i potentially releases an infinite sequence of MC instances, with consecutive instances being arrived at least T_i time units apart. Each sporadic task is represented as a 5-tuple:

$\tau_i^{£} = (£_i, T_i, D_i, E_i^2, E_i^1)$

where

- $£_i \in \{1,2\}$ represents the criticality level of task τ_i.
- $T_i \in R^+$ is the period (minimum arrival interval)
- $D_i \in R^+$ is the relative deadline. For an implicit deadline sporadic tasks, Di = Ti.
- $E_i^2 \in R^+$ is the estimated Worst-Case Execution Time (WCET) of workload $\tau_i^{£}$ in critical mode.
- $E_i^1 \in R^+$ is the estimated WCET of task $\tau_i^{£}$ in normal mode.

For LCT, $E_i^1 = E_i^2$ and for HCT $E_i^1 < E_i^2$. This is because the E_i^2 is very pessimistic than E_i^1 (to ensure timeliness guarantee).

OPERATING MODES OF THE SYSTEM

In a sporadic task model, each workload will have definite values of period Ti and criticality $£_i$. But, its completion deadline E_i is indeterminate; it is only discovered by executing an instance from the workload until it signals completion. If all workloads signal completion without overrunning E_i^1, the system has exhibited its normal mode. On the other hand, if any workload signals completion after overrunning E_i^1 but not exceeding E_i^2, then it has exhibited its critical mode behavior. If any workload executes for more than E_i^2, then the system has revealed erroneous behavior

Without loss of generality, we believe that all the cores enter into critical mode simultaneously and all incomplete instances from LCTs are canceled during the mode transition. The rationale behind this assumption is based on the fact that it is not essential to assure LCTs a high-level guarantee once a system exhibits its critical behavior. Task migration can be achieved as soon as the criticality switch commences. Once all the incomplete instances of HCT are finished execution, the system returns to normal mode while maintaining the schedulability of all necessary tasks.

UTILIZATION

Utilization denotes the overall fraction of an execution time demand of a system. It is used only to designate real-time recurrent (periodic or sporadic) workloads. Given a set of MC sporadic workload τ_i, the normal utilization of low-criticality task τ_i^1 in N-Mode is defined as:

$$U_i^1(\tau_i^1) = \frac{E_i^1}{T_i} \quad (1)$$

The critical utilization of high-criticality task τ_i^2 is defined as

$$U_i^2(\tau_i^2) = \frac{E_i^2}{T_i} \quad (2)$$

MC-SCHEDULABILITY

A task $\tau_i^{£}$ is MC-schedulable if and only if each

instance from $\tau_i^{£}$ is MC-schedulable. A set of MC sporadic workload τ_i is considered to be temporally correct (schedulable) by an algorithm ξ if and only if ξ will consistently satisfy all deadline constraints of τ_i to the certain degree of guarantee. The task set τ_i is deemed feasible if there exist some ξ such that τ_i is schedulable by ξ. The MC schedulability problem is known to be NP-hard. This rigorous result holds even in the extremely worst-case scenario such as all instances of the task have equal periods, and confidence level of each job is either 1 (low) or 2 (high).

CRITICALITY-AWARE PARTITIONED MC SCHEDULING

This paper concentrates on the problem of effective static-priority scheduling in an MC tasks on a multicore processor. An MC system behaves differently based on their execution mode. As stated above, the parameters of an admitted workload vary from one mode to another. This variation is not necessarily uniform between all cores since it hinges on the scheduling constraints of the workloads assigned to the core and the technique implemented by CA to estimate their WCET. So, the idle computational power existing for LCTs across cores does not necessarily modify by the same ratio when a criticality switch happens. To date, partitioning in an MC system is achieved such that the timing constraints of HCTs in C-Mode and the timing constraints of the LCTs in N-Mode are met simultaneously. This can be excessively conservative.

Many task-to-core mapping algorithms have been employed to the multicore scheduling of conventional real-time workloads and of late applied for MC applications (Kelly, *et al.*, 2011). Some of the most widely used approaches are BF, WF and FF. To increase the performance of partitioning heuristics when used to MC applications, we propose to exploit our Criticality-aware Partitioned Scheduling. Using this algorithm, a set of task is completely partitioned under the stable execution modes (N-Mode and C-Mode). Nonetheless, the bin-packing approaches used to map HCTs and LCTs are not necessarily the same. An LCT τ_i^1 in the system may have two dedicated cores Ψ_1 and Ψ_2. This technique circumvents the limitations of GS, especially, the necessity to exploit the conservative limits of GS, while being able to execute more tasks than PS as illustrated by the experiments in the following section.

In the context of operating mode, while the system remains to perform in a specified mode, all workloads are performed on their dedicated cores. But, when a criticality switch happens, workloads may transfer to other available cores. Task migration is achieved dynamically to provide effective utilization of resources in C-Mode. In order to maintain deterministic behavior of HCTs, they are not allowed to migrate. Only LCTs can transfer such that they are assured to receive a minimum execution level.

ILLUSTRATIVE EXAMPLE

In order to exemplify the tenet of our Criticality-aware Partitioned algorithm, we consider a sporadic task set comprising of 4 workloads to be scheduled onto two cores $\{\Psi_1,\Psi_2\}$ as depicted in Table 1. As we consider implicit-deadline workloads, the deadline of the workload is equal to its corresponding inter-task arrival time. Tasks are scheduled under Rate-Monotonic scheduling approach. In the case of workloads with equal inter-task arrival time, the workload with the lower index has greater priority. Without using PS algorithm, it is not feasible to schedule these workloads on the given cores.

In normal mode, tasks τ_1^2 and τ_2^1 are allocated to core Ψ_1, while τ_3^1 and τ_4^2 to Ψ_2. In N-Mode, the cumulative utilization of task τ_1^2 and τ_2^1 is $U_1^2 + U_2^1 = 0.30 + 0.15 = 0.45 \leq 1$ and the cumulative utilization of task τ_3^1 and τ_4^2 is $U_3^1+U_4^2 =0.25+0.30 = 0.55 \leq 1$. Hence, these tasks are scheduled across two cores without violating their timing correctness. In C-Mode, the utilization of task τ_1^2 is 0.90; hence τ_1^2 cannot be scheduled to a particular core with any other task. But, the cumulative utilization of the other three workloads would be $U_2^1 + U_3^1 + U_4^2 = 0.15+0.25+0.60 = 1$; and hence these three tasks are schedulable on a common core under C-mode. With CaPA, it is possible to assure the temporal correctness of the system by migrating tasks. In C-Mode, τ_2^1 transfers to Ψ_2 the other workloads continue on their designated core. (Fig. 1) illustrates the run-time execution behavior of this task allocation.

- **N-Mode (in an interval [0,26])**

The system starts its execution in N-Mode. In this mode, tasks τ_1^2 and τ_2^1 are allocated to core Ψ_1, while τ_3^1 and τ_4^2 to Ψ_2. At time t = 26, τ_1^2 executes for more

Table 1. A dual-criticality task set

Task	£	E_i^1	E_i^2	T_i	U_i^1	U_i^2
τ_1^2	2	6	18	20	0.30	0.90
τ_2^1	1	6	6	40	0.15	0.15
τ_3^1	1	10	10	40	0.25	0.25
τ_4^2	2	6	12	20	0.30	0.60

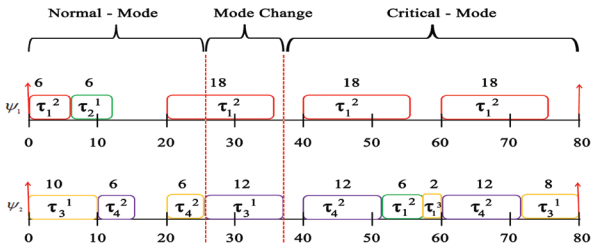

Fig. 1 Example system execution trace.

than its E_i^1 (i.e., 6 time units) and hence the system enters into C-Mode. At this time, τ_2^1 starts to transfer to Ψ_2.

- **Mode change (in an interval [26,38])**

All the active LCTs (i.e., τ_3^1 in our example) are discarded and no further tasks are permitted to perform for the full mode changing period.

- **C-Mode (in an interval ([38,80]:**

At t = 38, all incomplete instances finish and the system enters into C-Mode and τ_2^1 is executing on Ψ_2.

The Criticality-aware Partitioned algorithm (CaPA) (Algorithm 1) is developed to achieve efficient partitioning. We use two different partition techniques: NORMAL-partition is implemented in N-Mode and CRITICAL-partition for C-Mode. Our CaPA scheme divided into two parts: partitioning and optimization.

PARTITIONING

CaPA introduces two partitioning strategies (i.e., NORMAL-partition and CRITICAL-partition). This is realized in three steps: first allocate the HCTs, followed by LCTs in C-Mode, and finally LCTs in N-mode.

For each step, a task-to-core allocation policy is used to allocate tasks. Task-to-core allocation () in Algorithm 2 is used for allocating workloads in a particular mode based on the packing criteria Pack such as WF or BF. For the partition in C-Mode (i.e., Mode = 2), workloads are first ordered by non-increasing critical utilization and cores are ordered according to Pack. Then, workloads are allocated serially. Before a workload τ_i is allocated to a core, the schedulability of the core needs to be tested. It is only essential to test the schedulability in C-Mode by means of SCHED-CRITICAL (); for the partition in N-Mode (i.e., Mode = 1), workloads are ordered by decreasing U_i^1 and allocated serially after testing the schedulability of the core by means of SCHED-NORMAL-and- M_CHANGE ().

The algorithm CaPA can be implemented with any task-to-core allocation technique. By varying the task ordering principles, the packing criteria for HCTs (Pack 1 in Algorithm 1) and for LCTs (Pack 2), various algorithms can be developed. Simulation results revealed that using WF for partitioning HCTs yields the best results, as it allocates the HCTs across all the available cores uniformly leaving more freedom to partition the LCTs across all available cores. For LCTs, it is evident that the implementation of FF realizes higher schedulability for these workloads.

Algorithm 1:

CaPA (τ, Ψ, Pack1, Pack2)

1: $[\tau^2, \tau^1]$ = Split (τ)

2: Step: 1.a: allocate HCTs ▶Partitioning

3: If Task-to-core allocation (τ^2,Ψ,Pack1,2)==FALSE then

4: Return FAILURE

5: Endif

6: Step 1.b: allocate LCTs in C-Mode

7: If Task-to-core allocation (τ^1,Ψ,Pack1,2)==FALSE then

8: Return FAILURE

9: Endif

10: CRITICAL-partition = current partition

11: Step 1.c: allocate LCTs in N-Mode

12: DeAllocateTasks (τ^1)

13: If Task-to-core allocation (τ^1,Ψ,Pack2,1)==FALSE then

14: Return FAILURE

15: Endif

16: Step 2: optimizeallocation ▶ Optimization

17: MIG_TASK = tasks τ_i with Ψ_i^1 != Ψ_i^2 ordered by decreasing U_i^1

18: For all $\tau_i \in$ MIG_TASK do

19: If SCHED-NORMAL-and-M_CHANGE (τi,Ψ_i^2) then

20: Allocate (τ_i, Ψ_i^2)

21: MIG_TASK.update ()

22: Endif

23: Endfor

24: For all τ_i, $\tau_j \in$ MIG_TASK do

25: If Ψ_i^2== Ψ_j^1 then

26: If SCHED-NORMAL-and-M_CHANGE (τ_i, $\Psi_i^2 - \tau_j$) &&

SCHED-NORMAL-and-M_CHANGE (τ_j,$\Psi_i^1 - \tau_i$) then

27: Swap (τ_i, τ_j)

28: MIG_TASK.update ()

29: Endif

30: Endif

31: End for

32: NORMAL-partition = current partition

33: Return SUCCESS

OPTIMIZATION

This phase optimizes the partitioning achieved in the previous phase and decreases the switching overhead. This is accomplished by switching the allocation of some LCTs in N-Mode. The workloads that transfer during switching mode are put in the list MIG_TASK and arranged by non-increasing order of U_i^1. Then, for each workload τ_i in MIG_TASK, two attempts are made to transfer the workload: (i) reschedule the workload in N-Mode to the same core Ψ_i^2 to which it is allocated in the C-Mode. If this fails, one more effort is made to exchange workload τ_i with another transferring workload τ_j from Ψ_i^2 such that $\Psi_j^1 = \Psi_i^2$ but $\Psi_j^2 \neq \Psi_j^1$. To check the schedulability of the system, we use two functions: (i) SCHED-CRITICAL () used in the CRITICAL-partition; and (ii) SCHED-NORMAL-and-M_CHANGE () used in the NORMAL-partition.

Algorithm 2

Task-to-core allocation (τ, Ψ, Pack, Mode)

1: Sort (τ, DU)

2: Sort (Ψ, Pack)

3: For each $\tau i \in \tau$ do

4: For each $\Psi j \in \Psi$ do

5: If Mode == 2 then

6: Sched = SCHED-CRITICAL (τi, Ψj)

7: Else

8: Sched=SCHED-NORMAL-and- M_CHANGE (τi,Ψj)

9: Endif

10: If sched then

11: Allocate (τi, Ψj)

12: Sort (Ψ, Pack)

13: NextWorkload

14: Endif

15: Endfor

16: If τi not mapped to any core then

17: Return FAILURE

18: Endif

19: Endfor

20: Return SUCCESS

Function for NORMAL-partition

As we know that instances from LCTs are discarded during mode transition, the method used for schedulability analysis in AMC-rtb algorithm (Baruah, *et al.*, 2011; Bini and Buttazzo, 2005) can be reclaimed for our work. The function, SCHED-NORMAL-and-M_CHANGE () is implemented to test the schedulability in normal execution and switching mode.

Function for CRITICAL-partition

In this partition, the set of workload mapped to the core during CRITICAL-partition is considered. All the workloads (HCTs and LCTs) are tested using their C-Mode constraints using SCHED-CRITICAL (). The Worst Case Response Time (WCRT) for workload τ_i in CRITICAL-partition is calculated as:

$$R_i^2 = E_i^2 + \sum_{j \in h^2(i)} \left\lceil \frac{R_i^2}{T_j} \right\rceil \quad (3)$$

where h^2 (i) indicates the set of workloads which have greater priority than the active workload on the same core. We now proceed to evaluate our CaPA algorithm for the multicore processor. First, we evaluate various bin-packing approaches and assess their performance by implementing our CaPA on them. We will use the symbolization a/b to designate various algorithms where 'a' is the task-to-core allocation approach used for HCTs and 'b' the allocation approach for LCTs, where a, b $\in$ {WF, FF, BF}. For instance, WF/BF is the algorithm developed by implementing WF to HCTs and BF to LCTs.

EXPERIMENTAL EVALUATION

For our experiments, the period the workloads are arbitrarily selected from the set {10, 20, 50, 100, 200, 400, 500, 1000} ms. The Cfact for HCTs defining the ratio between E_i^2 and E_i^1 was arbitrarily chosen from 1 to 4. Deadlines were implicit. The

utilization bounds of workloads were fixed using the UUnifast (Bini and Buttazzo, 2005) method with the low (high)-utilization capped at 0.5 (0.85) per workload. The UUnifast creates workloads such that their cumulative U_i^1 is equal to 85% of the total processing capacity of the system. The parameter U_i^2 is calculated from other arbitrary factors (Cfact and T_i). Any task set with $U_i^2 > U_\lambda$ (where U_λ is the total processing capacity of the system) was dropped. The parameter E_i^1 of the workload is then obtained from the inter-task arrival time (T_i) and U_i^1. The completion deadline E_i^2 is derived from E_i^1 and Cfact. Unless otherwise stated, 50% of the workloads in the system are HCTs. Furthermore, each data point in the graphs is drawn from 200 workloads.

In our first experiment, we evaluate the bin-packing heuristics before implementing CaPA, which consist of 9 mixtures of approaches (i.e., WF/WF, WF/BF, WF/FF, BF/WF, BF/BF, BF/FF, FF/WF, FF/BF, FF/FF). Moreover, we also report the throughputs of the original task-to-core allocation strategies, which order workloads by decreasing utilization (DU) thus neglecting their level of importance. (Fig. 2) exhibits the ratio of schedulable tasks (acceptance ratio) against the normalized U_i^1. For this experiment, 50 tasks (25 LCT, 25 HCT) were scheduled on 4 cores.

The results in (Fig. 2) show that the mapping techniques that allocate HCTs using WF policy provide considerably better performance than the remaining ones. Among these, using FF for LCTs provides the best acceptance ratio. Interestingly, this achieves a similar result as prior work in (Rodriguez, *et al.*, 2013) for EDF-VD algorithm. The rest of this section focuses on WF/FF heuristic owing to its superior performance and exhibits substantial enhancements in the context of implementing the CaPA to it (WF/FF-CaPA). Exploiting CaPA on other approaches makes identical enhancements. But, those results are not presented in this paper by reason of space constraints.

Workload is then obtained from the inter-task arrival time (T_i) and U_i^1. The completion deadline E_i^2 is derived from E_i^1 and Cfact. Unless otherwise stated, 50% of the workloads in the system are HCTs. Furthermore, each data point in the graphs is drawn from 200 workloads.

In our first experiment, we evaluate the bin-packing heuristics before implementing CaPA, which consist of 9 mixtures of approaches (i.e., WF/WF, WF/BF, WF/FF, BF/WF, BF/BF, BF/FF, FF/WF, FF/BF, FF/FF). Moreover, we also report the throughputs of the original task-to-core allocation strategies, which order workloads by decreasing utilization (DU) thus neglecting their level of importance. (Fig. 2) exhibits the ratio of schedulable tasks (acceptance ratio) against the normalized U_i^1. For this experiment, 50 tasks (25 LCT, 25 HCT) were scheduled on 4 cores.

The results in (Fig. 2) show that the mapping techniques that allocate HCTs using WF policy provide considerably better performance than the remaining ones. Among these, using FF for LCTs provides the best acceptance ratio. Interestingly, this achieves a similar result as prior work in (Rodriguez, *et al.*, 2013) for EDF-VD algorithm. The rest of this section focuses on WF/FF heuristic owing to its superior performance and exhibits substantial enhancements in the context of implementing the CaPA to it (WF/FF-CaPA). Exploiting CaPA on other approaches makes identical enhancements. But, those results are not presented in this paper by reason of space constraints.

(Fig. 3) illustrates the enhancement gained by implementing CaPA on the WF/FF heuristic. This combination, WF/FF-CaPA algorithm (i.e., Algorithm 1 with Pack1=WF and Pack2 = FF) dispatches more tasks than the traditional WF/FF heuristic, particularly at higher utilizations. At utilizations of 0.8 and higher the WF/FF-CaPA improves the acceptance ratio of WF/FF by 23% increasing at utilizations of 0.9 and beyond to 31%. To analyze the predictability of CaPA at different parameters, the basic scheduling factors were varied in the following simulations. The normalized utilization was varied by changing the number of the workload from 10 to 200 while preserving the parameter $U_i^1 = 0.75$ per core.

(Fig. 4) illustrates the acceptance ratio of tasks against the total number of admitted tasks (a higher number of tasks implies lower average task utilizations). A close observation of the (Fig. 4) shows that WF/FF-CaPA increases the acceptance ratio of WF/FF by an

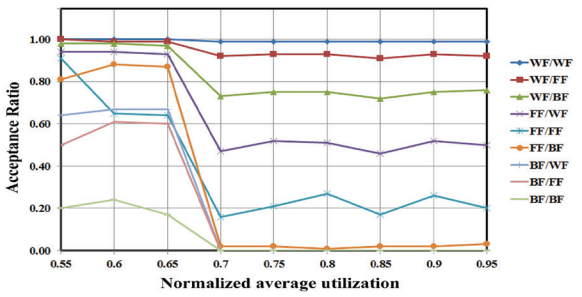

Fig. 2 Acceptance ratio at different Ui1 for various heuristics (before applying CaPA).

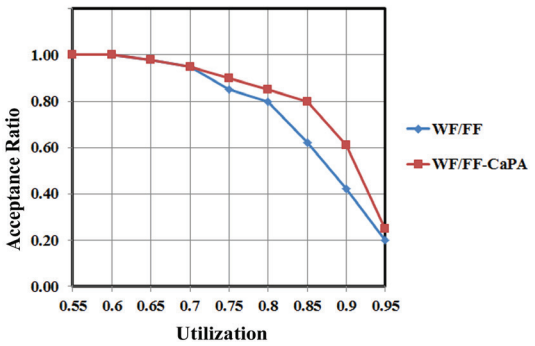

Fig. 3 Enhancement of the acceptance ratio at different U_i^1.

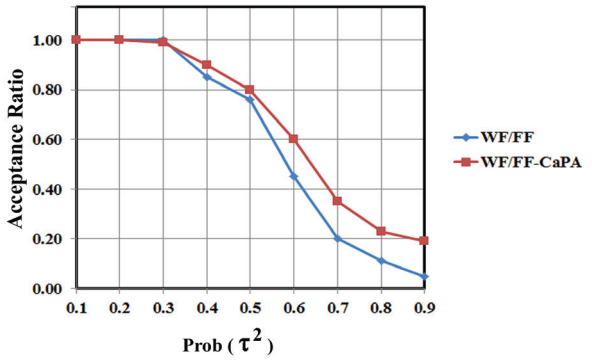

Fig. 4 Enhancement of the acceptance ratio for varying number of workloads.

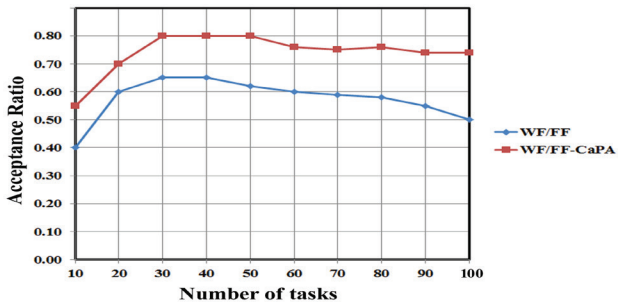

Fig. 5 Enhancement on acceptance ratio at different Prob (τ^2).

average of 24%. This ratio is increased for systems with a large number of workloads.

(Fig. 5) shows the impact of Prob (τ^2) on CaPA schedulability. Here, the parameter Prob (τ^2) represents the probability of a task to be a high-criticality task. The Prob (τ^2) is varied in the range [0.1 – 0.8] while preserving all other scheduling factors constant. For scheduling scenario with Prob (τ^2) ≤ 0.3, almost all tasks are schedulable. As the Prob (τ^2) increases, the acceptance ratios of both approaches reduce but WF/FF-CaPA performs better. The implementation of our algorithm in WF/FF packing heuristic (i.e., WF/FF-CaPA) schedules 73.3% of tasks whereas WF/FF schedules 54.6%.

CONCLUSION

There has been a growing research interest in the scheduling of MC workloads in multicore platforms. We investigate the static-priority partitioned scheduling on safety-critical multicore platforms in this document. We evaluate the performance of different task-to-core mapping approaches when applied to sporadic MC tasks. We also develop a Criticality-aware Partitioned Algorithm for multicore processors. Our proposed algorithm combines a partitioned scheduling approach with various mapping strategies to enable better utilization of system resources. CaPA is easily implemented with any task mapping heuristic and can always exhibit noticeable schedulability improvements on HCTs while delivering a certain level of timing assurances to LCTs. Future work includes extending CaPA to consider common platforms and run-time overheads (e.g., context switches and migrations).

REFERENCES

Anderson, J.H., Bud, V. and Devi, U.C. (2005). An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In: Proceedings of EUROMICRO Conference on Real-Time Systems (ECRTS). 199–208.

Andersson, B. (2008). Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In: Proceedings of ACM International Conference on Principles of Distributed Systems (OPODIS).5401 : 73-88.

Andersson, B., Baruah, S. and Jonsson, J. (2001). Static-priority scheduling on multiprocessors. In: Proceedings of 22nd Real-Time Systems Symposium (RTSS). IEEE.

Baruah, S., Burns, A. and Davis, R.I. (2011). Response-time analysis for mixed criticality systems. In: Proceedings of 32nd Real-Time Systems Symposium. 34-43.

Baruah, S., Chattopadhyay, B., Li, H. and Shin, I. (2014). Mixed-criticality scheduling on multiprocessors. Real-Time Systems. 50 : 142-177.

Bini, E. and Buttazzo, G. (2005). Measuring the performance of schedulability tests. Real-Time Systems. 30 : 129-154.

Burns, A. and Baruah, S. (2013). Towards a more practical model for mixed criticality systems. In: Proceedings of 1st Workshop on Mixed-Criticality Systems (collocated with RTSS).

Buttazzo, G.C., Lipari, G. and Abeni, L. (1998). Elastic task model for adaptive rate control. In: Proceedings of 19th IEEE Real-Time Systems Symposium. 286-295.

Gu, C., Guan, N., Deng, Q. and Yi, W. (2014). Partitioned mixed-criticality scheduling on multiprocessor platforms. In: Proceedings of Design. Automation and Test in Europe Conference and Exhibition (DATE).

Kelly, O.R., Aydin, H. and Zhao, B. (2011). On partitioned scheduling of fixed priority mixed-criticality task sets. In: Proceedings of 10th International Conference on Trust, Security and Privacy in Computing and Communications. 1051-1059.

Mollison, MS., Erickson, J.H., Anderson, B., Baruah, S K. and Scoredos, JA. (2010). Mixed-criticality real-time scheduling for multicore systems. In: Proceedings of 10th International Conference on Computer and Information Technology (CIT). 1864-1871.

Rodriguez, P., George, L. and Goossens, J. (2013). Multi-criteria evaluation of partitioned EDF-VD for mixed criticality systems upon identical processors. In: Workshop on Mixed Criticality Systems.

Su, H. and Zhu, D. (2013). An elastic mixed-criticality task model and its scheduling algorithm. In: Proceedings of Design. Automation and Test in Europe Conference & Exhibition (DATE).

Su, H., Zhu, D. and Moss´e, D. (2013). Scheduling algorithms for elastic mixed criticality tasks in multicore systems. In: Proceedings of 19th International Conference on Embedded and Real-Time Computing Systems and Applications. 352-357.

Yip, E., Kuo, M., Broman, D. and Roop, P. (2014). Relaxing the synchronous approach for mixed-criticality systems. In: Proceedings of 20th IEEE Real-Time and Embedded Technology and Application Symposium. 89-100.